Sonifying and Visualizing the Heartbeat of Evolving Software Systems

Carmen Armenti, Marco Raglianti, Michele Lanza
REVEAL @ Software Institute – USI, Lugano, Switzerland

***Abstract*—The lifecycle and development of software systems are strongly dependent on *time*: A critical dimension that must be considered when analyzing software evolution. Many visualization approaches have been proposed to support developers in analyzing software systems. Yet, most of these focus on static representations, which struggle to convey evolution *in time*, and leverage only vision. In contrast, hearing—although underutilized—is well suited for processing sequential information, making sound a powerful medium to convey changes chronologically.**

We present a multimodal approach, implemented in a tool named SONICSIGHT, that combines software *sonification* and visualization to analyze the development pace of software repositories interactively. A pulse synthesizer modulates its speed based on daily commits, while frequencies represent individual developers and their contribution activity. To support interpretation, the sonification is paired with a real-time interactive visual representation of software-related information. We illustrate such *sonified visualization* approach through case studies and discuss the underlying time model we employed, crucial for representing both sound and the temporal nature of software evolution.

***Index Terms*—visualization and sonification, multimodality, software evolution, sonified visualization**

I. INTRODUCTION

Software systems are intended to be designed to change. Over time, they evolve through modifications aimed at meeting new requirements, fixing defects, and adapting to the environment [1]. Almost all software engineering activities require program comprehension, which encompasses the techniques developers use to analyze and reason about software systems. Building this understanding demands a significant portion of developers' time [2], as they build mental models from the source code they read to make sense of the system [3]–[5]. One strategy for gaining insights into software systems is to examine their development history, which provides valuable context and reveals the motivation behind past changes [6]. Developers frequently refer to historical information to gain knowledge and ensure that their coding decisions remain consistent with the system's original design intent [7].

Although software history plays a crucial role in program comprehension, analyzing it remains challenging. The complexity of the domain stems from the heterogeneous and often cluttered nature of evolving software systems. Among the various approaches proposed to support program comprehension, software visualization offers one way to support developers in better understanding complex systems [8]–[12]. It aims to facilitate comprehension by employing visual aids and metaphors that make software structures more concrete and accessible, helping to reify abstract concepts [3], [13].

While traditional software visualization can assist developers, the majority of existing approaches lack an explicit representation of *time*, focusing on static snapshots of software systems. Such representations are effective for analyzing single states of a system, but fall short of conveying *how* the system has evolved in the past, or might evolve in the future.

Software, as part of an evolutionary process, has components that interconnect through time and metrics that evolve accordingly. Neglecting the temporal dimension in software representations can lead to the loss of crucial contextual information—particularly when it comes to detecting evolutionary patterns that, if represented statically, remain implicit or require developers to mentally reconstruct them [14]. Moreover, the lack of dynamic temporal representations can significantly hinder program comprehension, as developers rely heavily on historical context to make informed decisions during software maintenance and evolution tasks [6], [7], [15].

Although vision is the primary human sense, research shows that it does not always dominate. Hearing has been found to play a leading role in temporal processing [16], [17] as it is the most precise sense for temporal judgment [18], also enabling humans to handle multiple streams of information at once [19], [20]. As visual processing relies on *gestalt* formation recognition, so does the perception of auditory information [21], [22]. Yet, sound as a medium has not been leveraged as much as visual aids to support program comprehension tasks.

Using sound to convey information is known as *Sonification* [23]. Being sound rooted in time, sonification offers a natural way to convey temporal change while supporting the simultaneous perception of multiple information streams. Representing software evolution involves inherently dynamic phenomena and often requires analyzing multidimensional data. This makes sound *(i)* a valuable medium for conveying changes chronologically, apt to distill rich information while also revealing details, and *(ii)* a compelling complement to visual representations in the context of software evolution.

While metrics like the number of daily commits are not reliable indicators of developer productivity, they serve as objective measures of liveliness within software systems. Since systems vary in nature, each evolves at its own pace. In this work, we explore how the history of a software system can be characterized through its pulsing events: Periodic signals depending on commit-based activity that reflect its vitality, much like a heartbeat. To this end, we introduce the concept of *software heartbeat* as an indicator of the unique rhythm and intensity of a system's evolution.

 DOI: 10.1109/VISSOFT67405.2025.00009

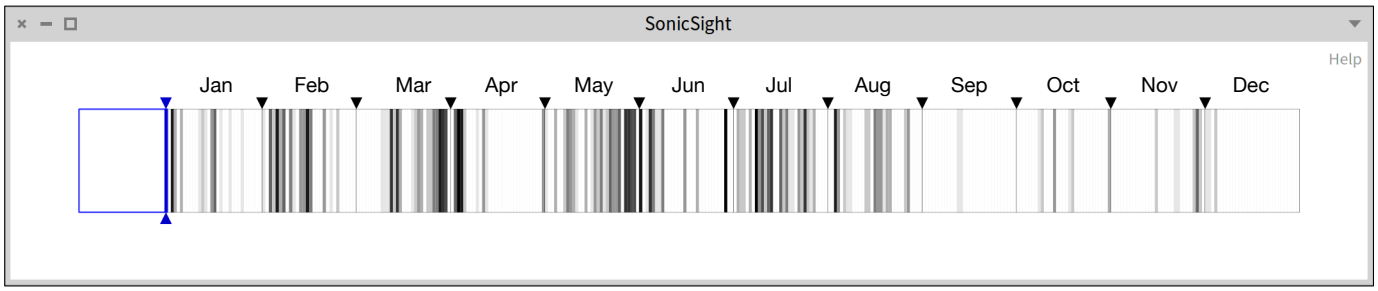

Fig. 1. Example Sonification of JetUML's evolution (2018): https://youtu.be/Z5-sl8RSVBw.

We propose a sonification approach, supported by a tool named SONICSIGHT, to represent the heartbeat of software repositories through the lens of developers' activity. Our approach is built on the idea that software systems behave as living organisms and that their (software) lifecycle exhibits lifelike patterns through development activity. We leverage two types of synthesizers: A pulse generator, which modulates its tempo in beats per minute (BPM) based on the daily repository's activity, and oscillators, which assign distinct frequencies to individual developers. The entanglement of recurring pulsing beats and frequencies serves as an auditory cue to reveal potential clusters of developer collaboration within the same time frame, and to assess how individual contributions align with the overall liveliness of the system. Crucially, the BPM indicator not only reflects activity, but it also conveys the intensity (strain?) that the evolution of a system may be experiencing: A high BPM can indicate a burst of productivity as well as a potentially excessive pace.

Prior research shows that sonification still requires coupling with some form of information visualization to convey the message effectively [24]. Building on this and emphasizing *multimodality*, we present SONICSIGHT, a multimodal analytics tool that pairs and synchronizes an auditory representation with an animated software visualization, supporting interpretation and providing richer contextual understanding.

By combining auditory and visual modalities, our approach enhances the perception of temporal patterns in software evolution, offering a novel means to see, hear, and interpret changes over time. Since time is central to the representation of sound, animation, and software evolution itself, our contributions include the time model that underpins this contextualization.

A. Example Sonification: The Base Layer

Figure 1 depicts a sonification of the JetUML system during 2018, thus spanning 12 months. The reader might be confused by the term "depicts a sonification". Whereas our focus is on the sonification, we also provide a depiction of what the reader can hear in this video: *https://youtu.be/Z5-sl8RSVBw*.

The period of 12 months is depicted as a horizontal sequence of vertical bars, and each bar denotes one day. Each day bar is colored according to daily development intensity, measured by the total number of commits on that date.

The top of the depiction features small triangle markers denoting the beginning (and end) of each month.

In the video, one can see a rectangle sliding over the visualization. The rectangle's right side is the "playhead", which shows which exact time frame is being sonified at any given moment. The width of the rectangle represents the duration of the sliding time window that we use to compute the average development intensity during that period, which in turn is mapped on the pulsing sound that one can hear, the "heartbeat" of the system. The shorter the considered time window, the narrower the playhead rectangle will be, and the more (and faster) the heartbeat will react to fluctuations in the number of commits. The heartbeat of the system serves not only the purpose of conveying the intensity of development it is undergoing at any moment, but is also sound-wise the rhythmic base layer over which we overlay other sounds (*e.g.,* single developer activity), as we discuss in the next sections.

What can be seen and heard in the example is that, in 2018, there were a couple of intense development periods, especially around the beginning of February, the 2nd half of March, and in May and July. The intense period in May was preceded by a very calm April. Also calm was the Fall of 2018 where, from the beginning of October to the middle of December, the development activity was nearly absent. Note that in this case, the sonification lasted 30 seconds. This is an arbitrary choice and seems adequate to represent a single year.

II. REIFYING TIME: THE SOFTWARE HEARTBEAT

A crucial element of software evolution analysis is the scale of software systems and their temporal dimension, as they can live on for several years, even decades. Providing comprehensive (and comprehensible) representations of software history that yield insights, regardless of the system's magnitude and lifespan, remains an open challenge.

The human understanding process necessitates cognition strategies and mental models [3], [25], [26]. Thus, given our purpose to represent software evolution dynamically at different time scales, an accurate modeling of time is necessary.

In Figure 2, we depict the approach we use to map development time on sonification time in SONICSIGHT and how we craft a sonified visualization of the lifetime of the repository. While we "compress" up to years in seconds, the playhead supports the synchronization between the visualization and the sonification to follow the evolution of the system.

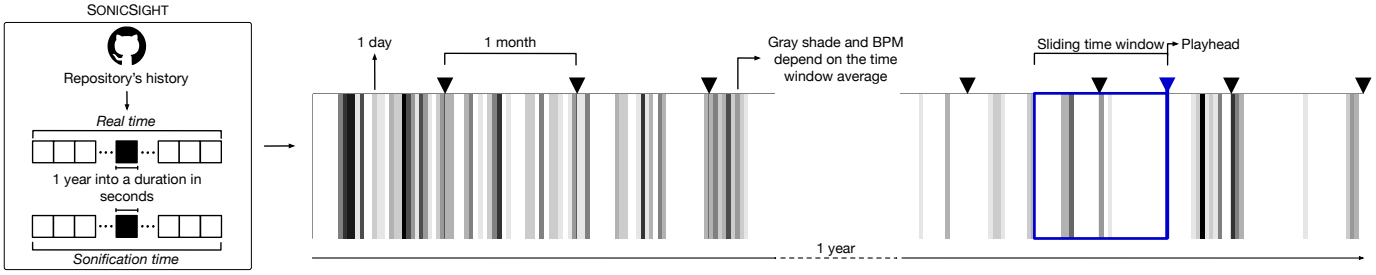

Fig. 2. The approach behind SONICSIGHT: Time mapping and sliding periods.

A. Mapping Time on Time

Version control systems (VCSs, *e.g., git*) track the evolution of software development by recording the history of changes, who made them, when, and why. The history of the system is represented as a graph of time-stamped events (commits).

Establishing a one-to-one mapping between the real-world timeframe and the reference timescale for the sonification is impractical. Realistic durations of repository lifespans are too long to be directly translated into usable soundtracks, as also observed by McIntosh *et al.* [27].

To address this, our time model converts *real time* into *sonification time*, a customizable duration that preserves the temporal properties of the original code changes (*i.e.,* the relative timing between events) while enabling the period of interest (*e.g.,* days, months, years of development) to be represented in a "digestible" amount of time.

The same applies to the extreme precision of VCSs in time-stamping events. While commits in *git* are recorded with the precision of milliseconds, our finer granularity is measured in days (the minimum time unit that makes sense to sonify for the evolution scales we are interested in). Hence, depending on the input duration, in the final sonification, each day will last for a specific number or fraction of seconds.

B. Carrying Temporal Information

Although carrying temporal information, VCSs fail to support software evolution analysis as they do not retain *sufficient* temporal information [28]. How VCSs document software changes favors the tracking of discrete time events (*e.g.,* when the modifications are committed), neglecting the actual period during which developers have worked to reach the point of committing their modifications to the codebase.

The number of commits in a day is often the result of multiple days of activity, and the daily development activity, when considered in isolation, offers limited insights into the overall progress or trends of a project. To tackle these two aspects, we adopt a *sliding time window* approach: While sounds and visual elements are still mapped with a daily granularity, the value associated with each day reflects the average activity over a customizable time window of n days. Instead of sonifying raw daily commits, the rolling average computed in the sliding window smooths out the *instantaneous* nature of time-stamped commits, resulting in a more comprehensive and representative sonification of the development pace.

C. Defining the Heartbeat

In the fields of music and digital audio, beats per minute (BPM) is a unit used to measure the *tempo* (or speed) of music by indicating how many beats are expected to occur in one minute. It is the core concept around which sound events align and synchronize to comply with a rhythmic structure. We created a mapping between a range of BPM values and the daily commit frequency, ensuring the preservation of code changes while still representing periods of inactivity.

No activity is represented as 30 BPM (1 pulse every 2 seconds), whereas the peak activity level is indicated by 150 BPM (2.5 pulses per second). These two values delineate the possible spectrum for our signal, but also point out two extreme conditions: An almost imperceptible and an excessively accelerated pulse, as can be observed in medical scenarios when they indicate problematic physical conditions.

Considering these two extremes, users can define a narrower inner tempo range to control the proportion of values treated as outliers. The narrower this user-defined range, the more values fall outside of it and are thus mapped directly to the minimum or maximum BPM (which has an audible non-linear "step" for special conditions). The number of daily commits within the inner range is linearly scaled between 60 and 120 BPM (1 to 2 pulses per second), allowing for a finer differentiation of typical commit-activity levels in a "healthy" repository.

Figure 3 compares three possible configurations for the inner range (0.1–4.0 (A), 0.5–3.5 (B), 1.0–3.0 (C)) in the example case of the JetUML project. The distribution of the average daily commits over 2015 (top), based on a 30-day sliding time window, shows the proportions of values falling within the inner range (blue, green, and gray, respectively) versus those considered outliers (red).

With the same colors, the lines in Figure 3 (bottom) show the mapped heartbeat for each inner range. The blue line, corresponding to the BPM for an inner range between 0.1 and 0.4, includes more values in the 60–120 BPM range, allowing for more gradual tempo transitions throughout the soundtrack. On the other hand, smaller inner ranges provide more amplitude for BPM variations to represent the activity of interest. The green and grey lines represent a progressively more hectic heartbeat rhythm, with larger outlier periods ((D) and (E)) and, in the middle range (60–120 BPM), more noticeable changes in activity.

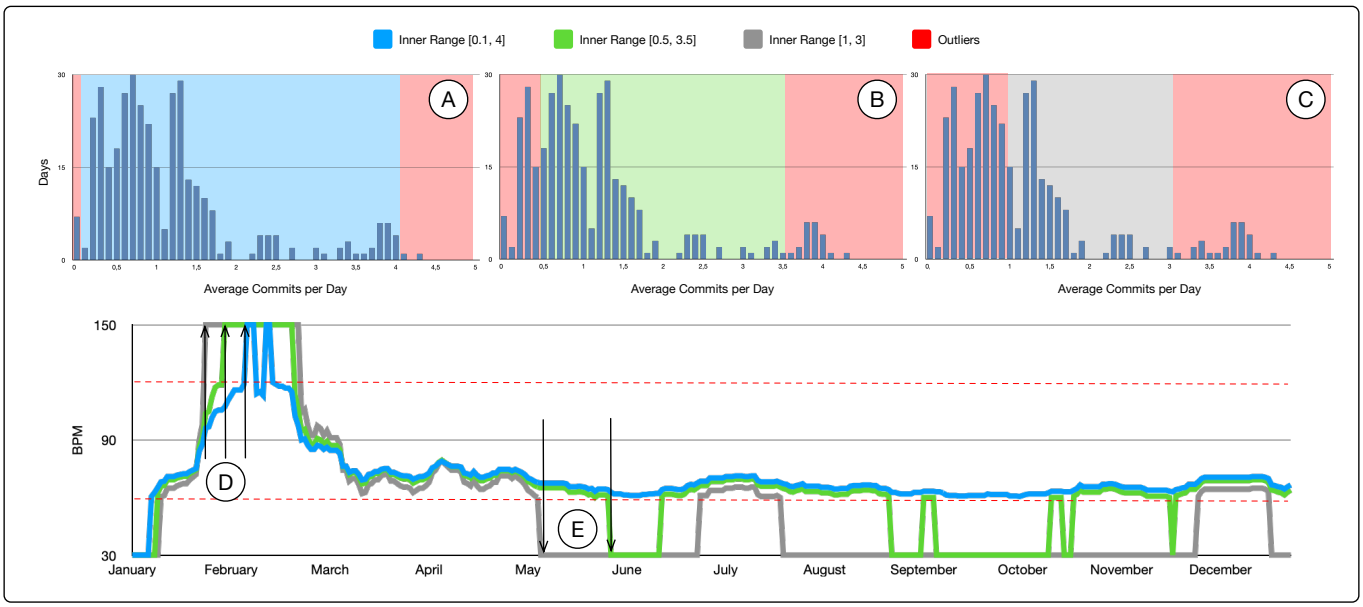

Fig. 3. At the top, three potential inner range configurations, tailored to the average daily commits distribution. At the bottom, BPM values for each range.

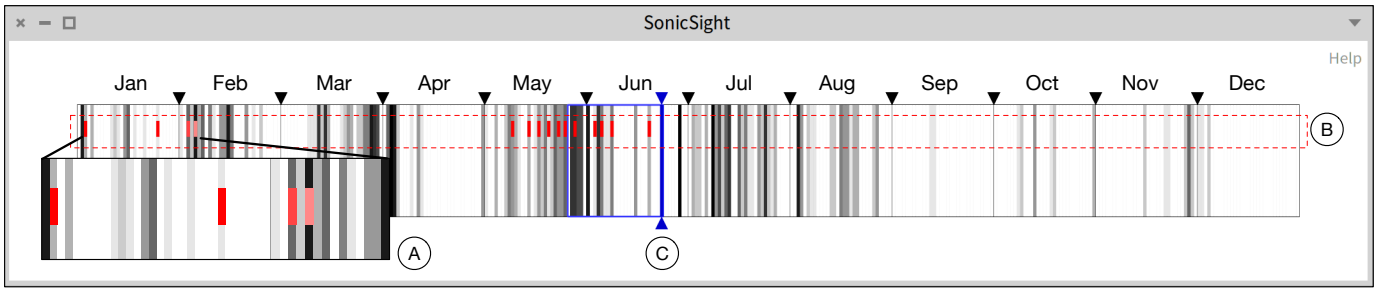

Fig. 4. Representation of individual developer activity in SONICSIGHT.

III. THE DEVELOPERS' SONIFICATION LAYER

The base audio layer forms the core *heartbeat* of the system, representing the overall commit activity. Since this activity is developer-driven, we enrich this layer with up to five parallel audio channels, each corresponding to one or more developers. Users may choose to highlight a top subset, up to four developers, while grouping the rest, or focus solely on the top subset (up to five developers).

Developers are ranked by the count of commits performed and subsequently mapped to pitches within a central frequency range (523–1,175 Hz), forming a harmonic chord: The top contributor is mapped to the root frequency, followed by intervals of a major third, fifth, major seventh, and ninth.

Visually, developers are represented by colored ticks, and the saturation of the color indicates their level of participation in the cumulative activity for a given day (Figure 4 (A)). Each developer is mapped to a horizontal slice of the heartbeat track ((B)), sorted from top to bottom, from the most to the least active. Developers appear in the animation whenever the playhead reaches a day in which they contributed ((C)).

The playhead's position advances in sync with the playback time: An abstraction of the project's real timeline.

This synchronization is driven by a user-defined "update frequency" which determines how often the position of the playhead and the BPM are updated. Typically, the shorter the update frequency, the smoother the animation.

For the sonification, instead, changing the BPM too often can result in distracting auditory artifacts (*e.g.,* interrupted heartbeats). For these reasons, we set an update frequency of 0.2 seconds, resulting in a smooth sonified visualization, even for high compression rates (*e.g.,* years in tens of seconds).

At each update, the playhead's position reflects a specific date. The colored ticks appear during sonification to mark these moments in the timeline. Similarly, in the audio channel, each added tick corresponds to an emitted sound to signify the activity of a specific developer.

In the examples presented so far, SONICSIGHT's soundtracks compress a year of development into a 30-second sonification, thus each month is represented by 2.5 seconds. Every 0.2 seconds, the playhead updates the BPM pulse and leads to the sonification and rendering of the active developers on the background heartbeat of the overall repository activity, effectively translating temporal development data into a complex, layered, and synchronized audio-visual rhythm.

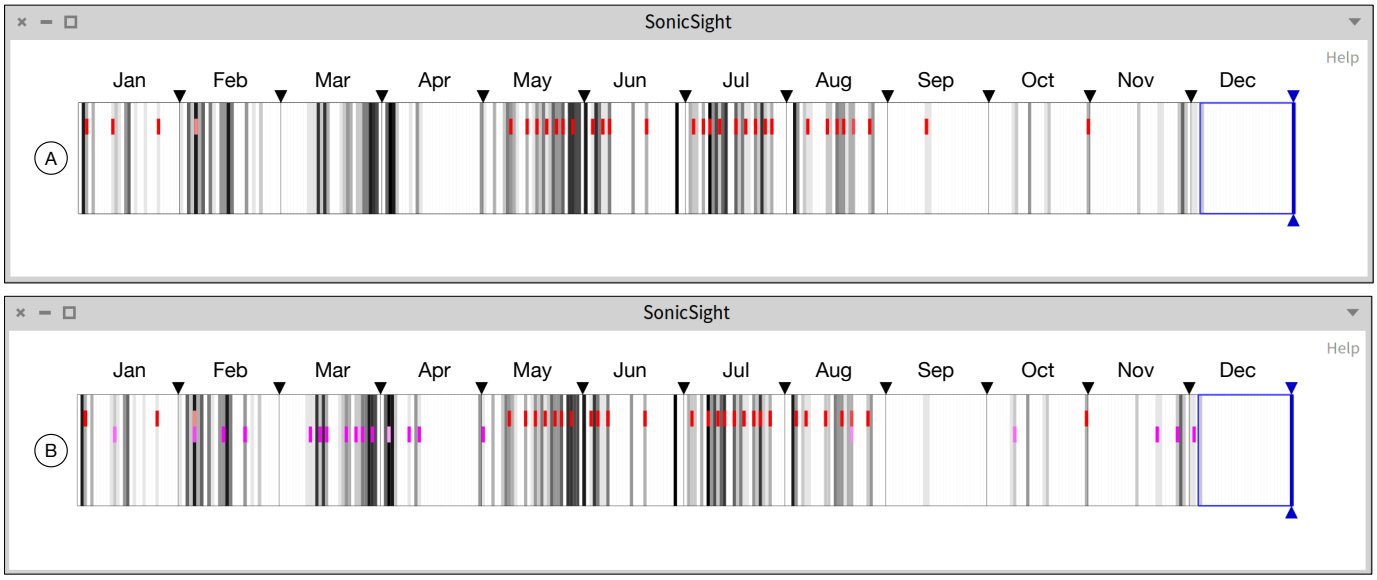

Fig. 5. JetUML's developers' activity Sonification (2018): A https://youtu.be/rzrw6X1Z7SM and B https://youtu.be/rgpRvwMaWXw.

We present two examples of the additive sonification layers. Given the limitations of Figure 5 to convey the result, we encourage the reader to hear an example of the progressive addition of layers to the sonification. The videos are here: *https://youtu.be/rzrw6X1Z7SM*, *https://youtu.be/rgpRvwMaWXw*.

A. The Main Developer

First, we add a single developer, the main driver of JetUML's evolution (Figure 5 (A)). As we can see and hear, Developer 1 does not participate in all intense activity periods, but seems mostly active between May and August 2018. The rest of the year, they only performed rare commits in January, February, and at the end of October.

Upon closer manual inspection,[1] those rare commits mostly concern occasional documentation, code refactoring, and small bug fixes. In contrast, during the central period, the activity reflects a more collaborative environment: Commit messages include issue tags, and there are more merge commits. This may be the result of a similarly intense activity by the other developers, which we can highlight by adding more layers.

B. The Main Developer and "the Others"

Layered over the heartbeat that drives the rhythm of activity in 2018, two distinct developer sonification layers are present (Figure 5 (B)): One line representing the main developer (red) and another one for all the others (magenta). When listening to the soundtrack, the addition of a second sound layer is perceptible. The auditory proximity of these two sounds reflects the relative ranking of the developers, where the main developer is represented individually, while all the others are grouped into a collective voice (from a single synthesizer).

[1]Which can be performed directly in SonicSight by inspecting the commits in the fully reified repository activity.

With this sonified visualization, we confirm our previous intuition. By observing and listening to the enriched second layer, one can perceive an alternation in activity between the main developer and the others, accompanied by an increase in BPM in the middle of their respective periods of activity. The resulting sonification, reinforced by the visual cues, reveals that the contributions of the main developer and the group are indeed temporally complementary.

Focusing for a moment on the sonification only, we can hear how it begins and ends with an alternation of two frequencies (corresponding to the musical pitches C4 and E4). The central section, instead, evolves into a dialogue between two voices: The first, representing all other developers, is later joined by a second, representing the main developer, whose activity becomes prominent at the beginning of May.

Between seconds 3 and 20 of the sonification, the heartbeat reaches its highest BPM and developer-related frequencies are played more often. These seconds correspond to the most substantial development period of the year, led by Developer 1 and framed by quieter phases with less "sonic density".

A manual inspection of the commits reveals that this central development phase was preceded by corrective maintenance (*e.g.,* fixing copyrights, removing classes) and adaptive change tasks, which appear to lay the groundwork for the subsequent phase of new features development beginning in February.

The sonification also captures collaborative dynamics (*e.g.,* early February) and marks transitions between contributors, around the beginning of May. This shift in auditory patterns also reflects a change in the main developer's focus: Their early commits differ from those made during the central phase of development, as previously observed in Section III-A.

Finally, for the sake of illustration, if the sonification time is set to 3 minutes, the sonified visualization is more fine-grained, slower-paced, and individual daily commits are audible.

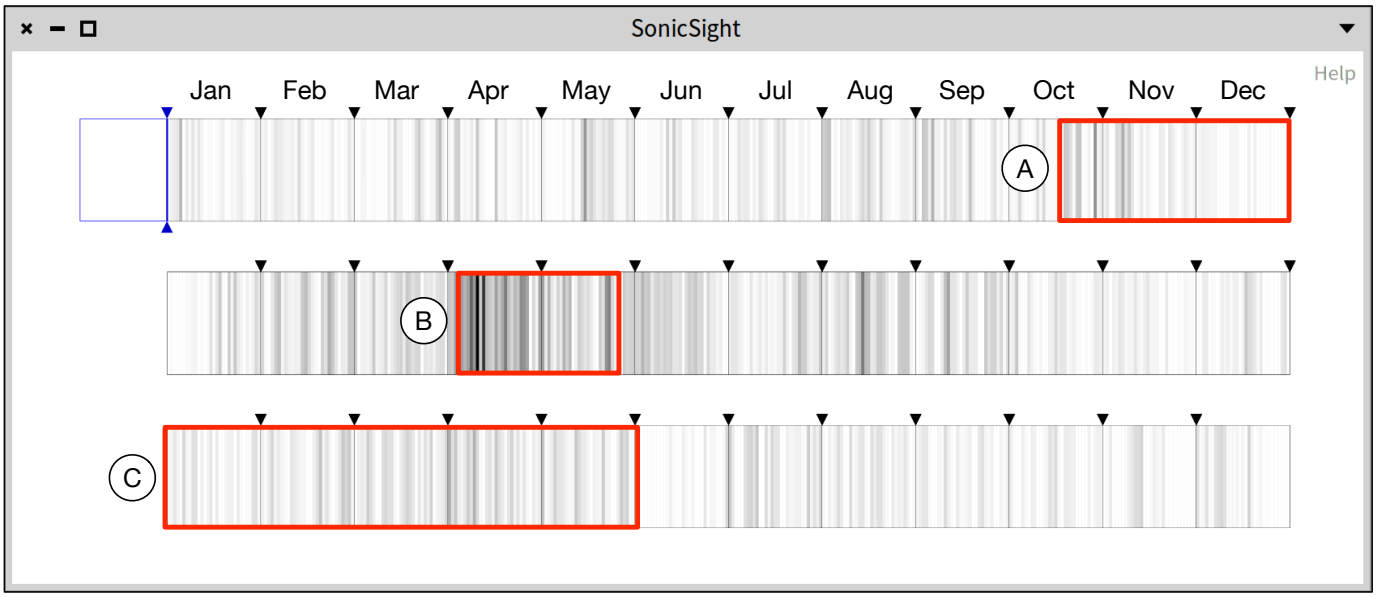

Fig. 6. Representation of 3 years sonification in SONICSIGHT: React from 2018 to 2020 (https://youtu.be/z0tC3LeQvyQ).

IV. MULTIMODAL ANALYSIS OF SOFTWARE EVOLUTION: CASE STUDIES OF SONIFIED VISUALIZATION

To evaluate the applicability of our approach, we present a series of case studies based on GitHub software repositories. These case studies aim to *(i)* illustrate how sonification can render software changes over time more understandable, and *(ii)* assess how different sonification strategies can convey different levels of detail and reveal collaboration patterns among developers. A key advantage shared across all cases is the ability of our approach to manipulate time. On the one hand, time can be compressed to provide an overview of *years* of development; on the other, it can be expanded to analyze shorter periods in detail, allowing listeners to perceive fine-grained changes, complemented by the synchronized visuals.

A. A Slow-Paced Software Heartbeat

In Figure 6, we show the evolution of the React software repository, Facebook's JavaScript framework for building web-based user interfaces. We analyze three years, from 2018 to the end of 2020, counting 4,229 commits. The sonification is available at: *https://youtu.be/z0tC3LeQvyQ*.

We use a sliding window of 30 days, and the total duration is 72 seconds. Each month of development is condensed into 2 seconds of audio. The distribution of the average number of commits per day made it relatively straightforward to determine the values for the inner tempo range.

As illustrated in Figure 7, over the three years, the development activity generally hovered between 1 and 3 commits per day, with occasional peaks reaching 15–18 commits, as well as completely inactive days (0 commits). Based on this pattern, we defined the inner range between 1 and 12 commits per day, mapping it to a tempo range of 60 to 120 BPM.

React's baseline development pace slows down in November and December 2018 (Figure 6 (A)).

Activity picks up again starting mid-January 2019. The rhythm begins to intensify and reaches a peak in April 2019 ((B)), where the heartbeat reaches the corresponding peak of 150 BPM, sustained for slightly more than a month. Then, the visualization seems to show a less intense daily activity (*i.e.,* less saturated bars) but a similarly active development pace. The sonification, instead, clearly represents a decrease in the pace, with its tempo slowing down in a perceptible way.

Towards the end of the year, the rhythm returns to its more familiar pattern, before rising in the first half of 2020 ((C)), to go back again to its baseline pulse until the end of the year.

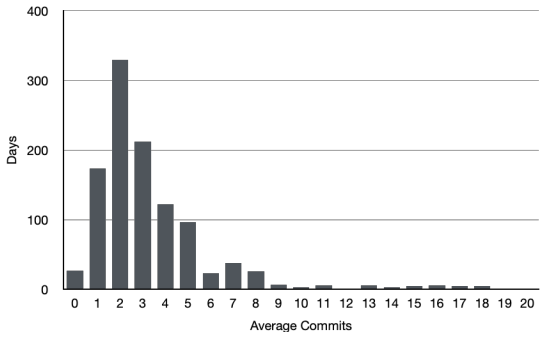

Fig. 7. Daily commits average from 2018 to 2020 in React.

Listening to the sonification produced from SONICSIGHT, the development activity is perceived from the outset as slow-paced and relatively steady, with perceptible increases in rhythm during distinct periods. A closer analysis reveals that the increases in activity are primarily driven by merge commits from branches and pull requests. Thus, the periods of heightened rhythm in the sonification not only reflect increased development activity but also signal integration phases in the project's workflow.

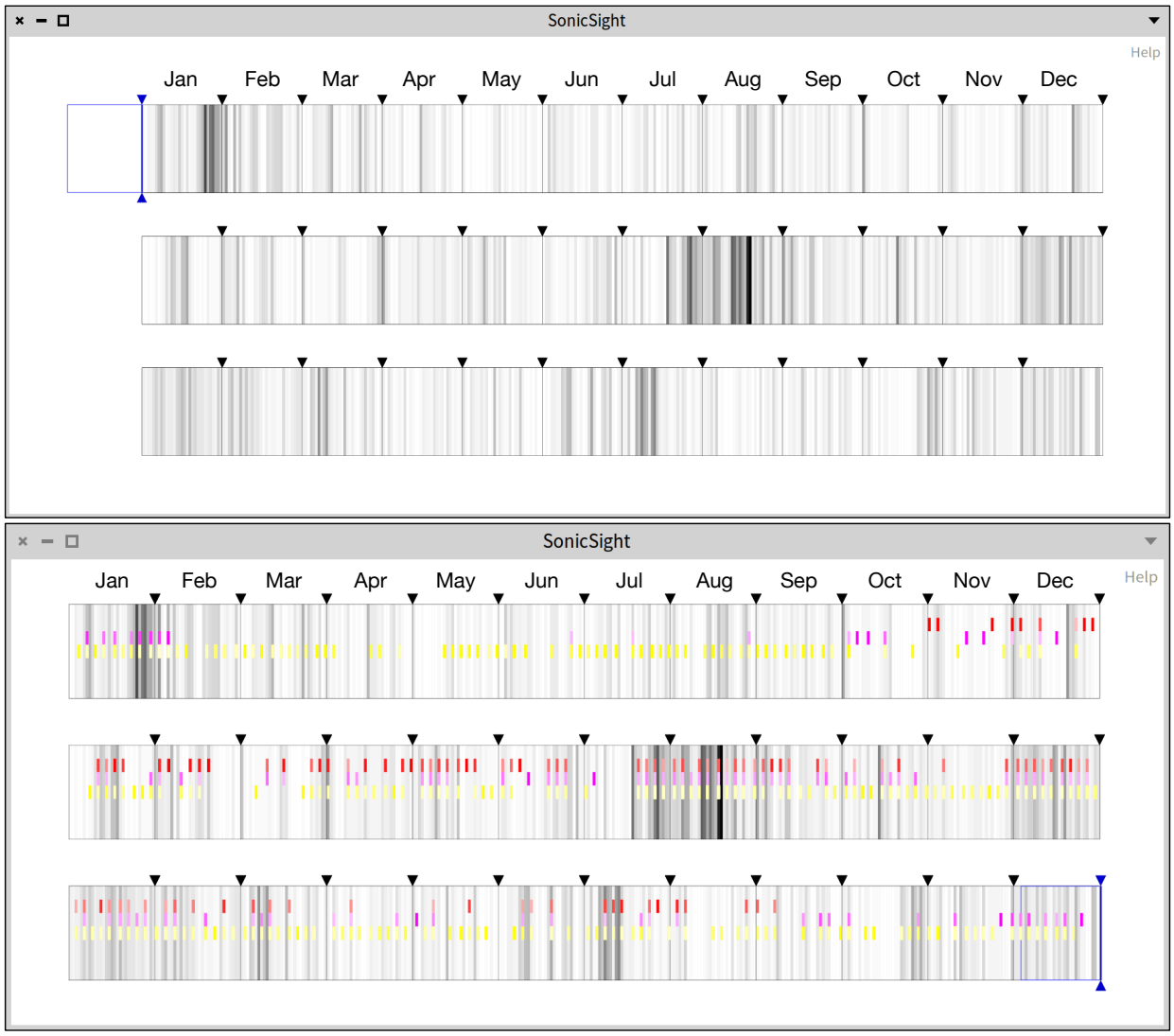

Fig. 8. Sonification layer 1 (top, https://youtu.be/4RF5V7lCgVk) and 2 (bottom, https://youtu.be/nfNP5ogc6ag) in SONICSIGHT: Bootstrap from 2012 to 2014.

B. How does Bootstrap and its Developers Sound?

Figure 8 is the depiction of the 2-layer sonification of 3 years (2012–2014) from the Bootstrap project on GitHub. It counts 10,244 commits. Similarly to the previous case study, its duration is set to 72 seconds, and we use a 30-day sliding time window. The sonification layers are here: *https://youtu.be/4RF5V7lCgVk*, *https://youtu.be/nfNP5ogc6ag*.

The distribution of commits per day (Figure 9) concentrates most of the values between 1 and 10, but also presents a secondary peak of increased pace around 14 (we hypothesize representing a specific crunch mode period). This observation led us to pick 1 and 12 as the minimum and maximum values of the inner range for this sonification configuration.

We layered the heartbeat with three oscillators. The first two (base pitch and third interval) represent the top 2 developers (red and magenta, Figure 8), while the last (yellow) describes the activity of *all* the others. It is worth noting that the third oscillator sonifies the cumulative activity of 670 people.

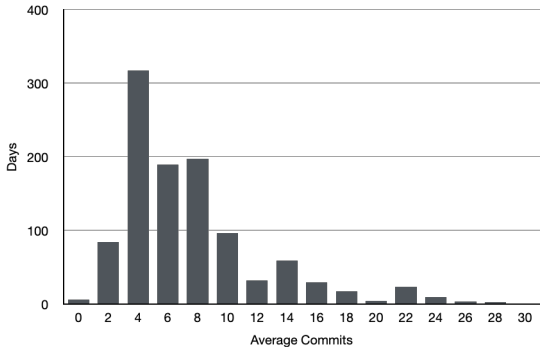

Fig. 9. Daily commits average from 2012 to 2014 in Bootstrap.

The heartbeat describes an alternating rhythm that presents peaks at the beginning of 2012, from mid-2013 until the beginning of 2014, and then again in the middle of 2014.

Listening to the sonification layer with developers, it seems that the increase in pulse is correlated with the addition of new contributors to the ongoing development activity. At the same time, we can hear in specific periods (*e.g.,* from September to December 2012) an alternation between the main drivers of the project (Developer 1 and 2) and the rest of the community, which acts in preparation or response to the main activity in a pattern. During fast-paced development stretches, instead, the different actors play in unison.

The year 2013 begins with an intense rhythm, as the second most active developer alternates with the rest of the community, contributing simultaneously until mid-February. Interestingly, when the first developer takes a break, the community also slows down its activity, and the heartbeat drops consequently.[2] Finally, when the main developers disappear (or their contribution becomes sporadic) and only the group of others remains, the rhythm settles in what looks and sounds like a "maintenance mode" (from February to May, 2014). The entrance and exit of the main developers strongly modulate rhythmic changes both in the underlying heartbeat and in the developer sonification layer.

Analyzing the contents of the commits, the first two developers are very likely the same person. For developer disambiguation, we relied only on the author information in each *git* commit (*e.g.,* username, email). We are aware of the issues related to identity ambiguity and acknowledge that relying on *git*'s author metadata does not necessarily provide accurate distinctions between real developers, with aliases, such as the same author with different emails [29]–[31]. However, our goal is to represent what *git* suggests should be inferred from the imperfect and noisy information it stores.

This leads to a reflection on the group of developers that is consistently present from the very first moment of the sonification until the end. One might argue that this is to be expected, given that the group consists of 670 individuals.

Nevertheless, the sonification emphasizes how collective activity can maintain a steady presence across time, even as individual contributors come and go.

C. Sonifying 9 Years: The VSCode Case

The final case study addresses a long sonification of almost a decade of a very active repository. It pertains to 9 years of history of the VSCode project on GitHub, during which it has accumulated over 120k commits, averaging about 39 daily commits with a 30-day sliding window (Figure 10). In this case, the duration of the sonification is set in a way that each month lasts 4 seconds: The final sonification lasts 7 minutes.

Our goal in this instance is to have an extended—therefore more gradual and fine-grained—sonification, allowing for the perception of subtler fluctuations in the inner range, which we map from the interval of [22–58] commits per day. Although the maximum value lies near the right tail of the distribution, we choose to include it, except for a few rightmost outliers.

[2]This apparent logic conclusion gives information instead on the absence of other major players that influenced the development pace of Bootstrap, at least in the considered time window.

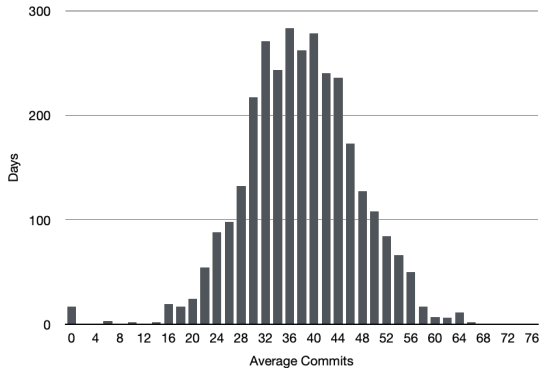

Fig. 10. Daily commits average from 2015 to 2024 in VSCode.

We believe this accelerated pace—in a repository like VSCode, and considering the shape of the distribution—still represents an integral part of the *normal* activity during a concerted phase, rather than a pathological condition.

Figure 11 depicts the whole heartbeat shape and, for the last time in this paper, we invite the reader to overcome the limitations of this medium and hear the sonification before continuing the reading. The video of the sonified visualization can be found here: *https://youtu.be/YPWt95ROd9w*.

The rhythm of the repository is steady, with a heartbeat that stays consistently below the average of 90 BPM, relatively slow paced at the very beginning (the average BPM was 77.6 in the first year). The first two years of the project show a recurring slowdown every 10 to 12 seconds, which corresponds to a period of 2.5 to 3 months. The final slowdown one can hear at the end of almost every year is likely related to the Christmas and end-of-year holidays.

Notably, 2018 seems to have been a period of peak activity. The rhythm is intense, with only a slight slowdown in the middle of the year, between June and August. In contrast, 2019 exhibits an even more significant slowdown at the end of April and the beginning of May, before returning to a high rhythm leading up to the Christmas break. The acceleration is anticipated and the slowdown delayed, though, sounding like expectation building up and releasing all of a sudden.

Unlike in previous years, 2020 begins with a sudden and intense rhythm that remains steady until the summer break (July–October). Something changed the pace of development in 2020 and, although easy to say a posteriori, the sonification represents CoViD times as a more constant pace that breaks the 3-months cycle we could hear in the previous years and anticipating a trend that will settle in 2022.[3]

The activity pattern in 2021 follows a similar dynamic, with a firm first half of the year, peaking in April 2021: The moment with the highest BPM across the timeline. From April 2022 onward, the overall activity becomes more stable and maintains this consistency through the end of the period, with the only exception being the final Christmas break in 2023.

[3]Far from implying causality, we still find that the emerging characteristics of the sonification result in informative insights across a large range of scales, if the sonification parameters are chosen according to the intended task.

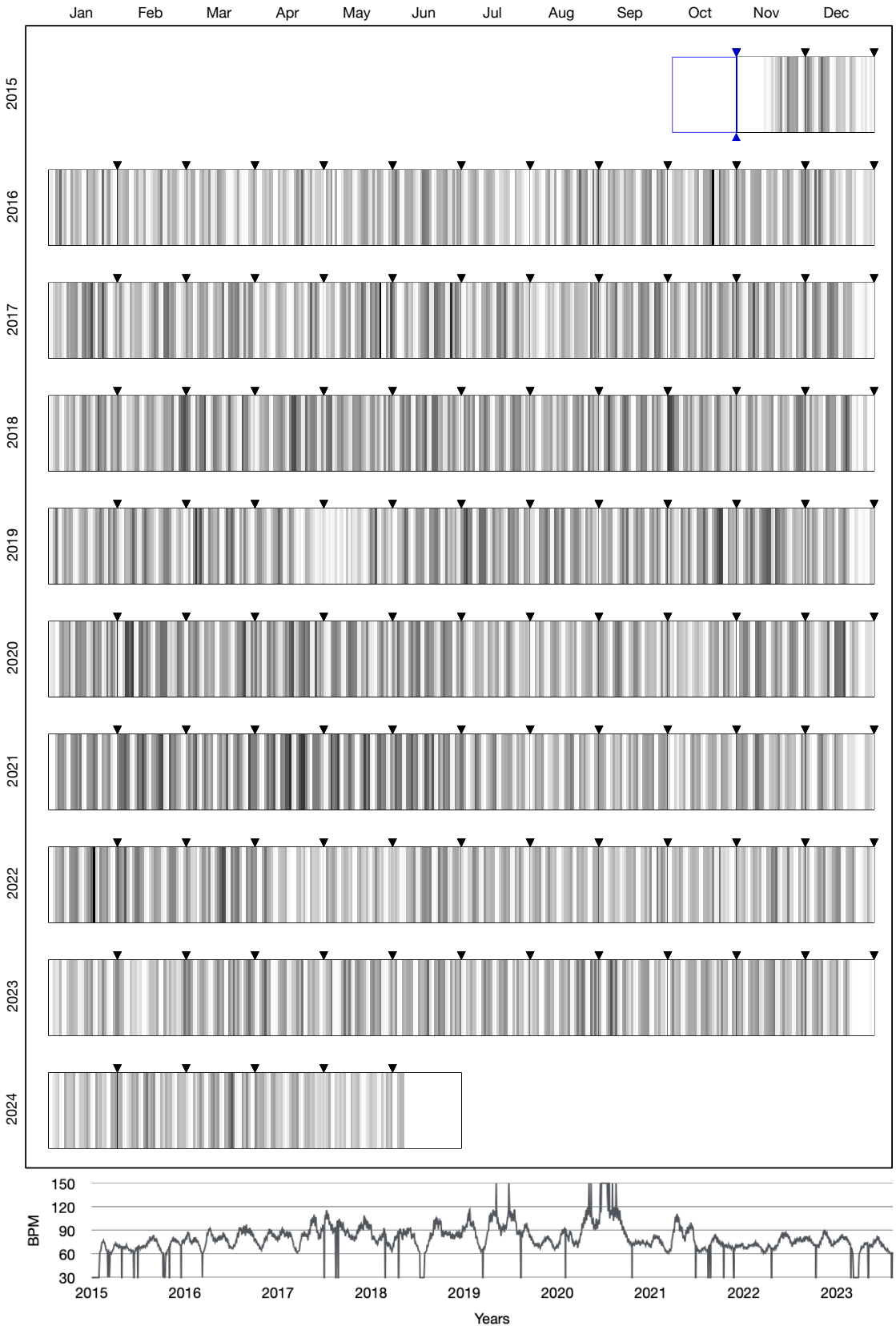

Fig. 11. 9 years sonification in SONICSIGHT (top) and BPM line chart (bottom): VSCode from 2015 to 2024 (https://youtu.be/YPWt95ROd9w).

V. Discussion

Time plays a fundamental role in analyzing the evolution of software systems. The ability to master time is a prerequisite for sonification and animation research. Sonification time as a one-to-one mapping with development time is impractical, with long pauses spent waiting to hear and observe something. What we implemented is meant to manipulate and bend time, compressing and dilating it dynamically.

Some might argue that combining two media could be distracting rather than helpful. Alas, although vision is the most used human sense, hearing is the most precise. It can detect subtle deviations that the eyes may miss [32]. Combining visual and auditory channels offers a complementary *multimodal* approach to enhance understanding, mitigating single-mode limitations. To illustrate this point, we use SonicSight's sonified visualization. Some adjacent shades of gray may be hard to distinguish, yet, when we combine sight and hearing, those differences are either reinforced or discriminated by hearing: Auditory cues convey changes in rhythm that may not be as easily distinguishable through vision alone.

Visual parameters also need to be controlled, just as we control sound parameters. We can manipulate the color scheme and saturation ranges in SonicSight. Shades of gray, for example, can be adapted to reflect the local or global maximum number of commits, according to the considered development period or fixed to constant values. The desired emphasis dictates such choices, similarly to how we handle inner ranges.

Interaction is another crucial ingredient to obtain *control*. A "movie-like" representation that can only be played or paused (*e.g.,* Gource [33]), while beautifully crafted, fails in supporting interactive system evolution analysis. To be effective, it is essential to empower users to interact with the model's entities, encouraging active inspection and exploration.

VI. Related Work

Software Visualization and Animation. Software visualization simplifies complex software-related concepts to support program comprehension [3], [4], [13], [34], [35]. Revision Towers represents file-level change information [36]. RepoGrams provides a metric-based visualization model to understand the evolution of metrics in a project's history [37]. 2D matrix visualizations improved software evolution comprehension [10], [38], [39]. Aghajani *et al.* proposed the Code Time Machine, a plugin that leverages visualization techniques and meta-information from VCS to represent files' history [40]. Kim *et al.* presented an interactive visual analytics system to represent a *git* graph of development history [41]. EvoLens combines structural and temporal views for evolutionary data exploration [42]. The Evolution Radar visualizes logical coupling at module and file levels [43]. The *city metaphor* depicts packages as districts, classes as buildings, supporting reverse engineering, comprehension, and evolution tasks [44], [45]. Dynamic visualizations help with processes [46] by factoring in temporal information via animations, conveying changes and object identities over time, which enables tracking [14] and spotting structural decay [47], [48].

Animations have been employed for analytical [36], [49], [50] and educational purposes [51], [52], depicting the inherent dynamics of software. In animated views, interaction allows the extraction of information on demand [14], [53], [54].

We are not the only proponents of mixing visualization and sonification. Their combination is more efficient [19], [55], [56] and can facilitate developers' comprehension [57], [58].

Sonification. The origins of sonification can be traced back to 1954 when Pollack and Ficks showed that the multi-dimensionality of sonification outperformed unidimensional displays [16]. Sonification can be used when vision is already engaged in comprehension tasks [59], or to simplify visualizations when they become overly complex [58], [60]. InfoSound uses music and sound effects to describe application events that are difficult to visually detect [61]. LogoMedia sonifies program behavior by mapping program events to sound, allowing one to *hear program execution* [62]. CAITLIN assists programmers in debugging tasks [63]. While North *et al.*, through GitSonifier, and McIntosh *et al.* focused on augmenting comprehension of version histories with sound [27], [64], Berman and Gallagher developed a sonification mapping to describe the low-level architecture of Java programs, showing its usefulness in static program comprehension [65].

VII. Conclusion

We presented sonified visualizations, a multi-modal approach to visualize through animations augmented with sonification, the evolution of software systems. One might be tempted to dismiss sonification as yet another channel through which one can encode data, such as mapping the number of commits onto musical pitches. However, as we progress with our research, we are learning that hearing is a fascinating, yet complex, sense. It obeys certain rules that need to be understood, very much like visualization relies on the rules of pre-attentive processing, in the form of *Gestalt* principles. Sonification heavily relies on a set of principles that we are uncovering as we go. One of the contributions of our investigations is the conception of a "base rhythmic layer" mimicking the heartbeat, which creates an intuitive and informative scaffolding on which we can overlay other consonant or dissonant sonifications. Another result is the need to treat "time" as a first-class citizen, and to make it malleable.

This necessity stems not only from the suboptimal way in which evolutionary information about software is persisted, but also from the fact that sonification is primarily time-bound. We manipulate time to make specific information emerge.

We illustrated our approach on several systems, highlighting the fact that well-designed sonifications combined with simple animations can indeed transmit multimodal and complementary insights about evolving software systems, which would otherwise be difficult to come by. In that sense, you will hear more about this in the future.

▶ ***All Videos Playlist:*** *https://tinyurl.com/2rfs2xhf*.

Acknowledgments: Work supported by the Swiss National Science Foundation, project "FORCE" (SNF 232141).

REFERENCES

[1] L. A. Belady and M. Lehman, *Program evolution: Processes of software change*. Academic Press, 1985.

[2] R. Minelli, A. Mocci, and M. Lanza, "I know what you did last summer–An investigation of how developers spend their time," in *Proceedings of ICPC 2015 (23rd International conference on Program Comprehension)*. IEEE, 2015, pp. 25–35.

[3] M. D. Storey, F. D. Fracchia, and H. A. Müller, "Cognitive design elements to support the construction of a mental model during software exploration," *Journal of Systems and Software*, vol. 44, no. 3, pp. 171–185, 1999.

[4] M.-A. Storey, "Theories, methods and tools in program comprehension: Past, present and future," in *Proceedings of IWPC 2005 (13th International Workshop on Program Comprehension)*. IEEE, 2005, pp. 181–191.

[5] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: A study of developer work habits," in *Proceedings of ICSE 2006 (International Conference on Software Engineering)*. ACM, 2006, pp. 492–501.

[6] T. D. LaToza and B. A. Myers, "Hard-to-answer questions about code," in *Proceedings of PLATEAU 2010 (2nd Workshop on Evaluation and Usability of Programming Languages and Tools)*. ACM, 2010.

[7] M. Codoban, S. S. Ragavan, D. Dig, and B. Bailey, "Software history under the lens: A study on why and how developers examine it," in *Proceedings of ICSME 2015 (31st International Conference on Software Maintenance and Evolution)*. IEEE, 2015, pp. 1–10.

[8] C. V. Alexandru, S. Proksch, P. Behnamghader, and H. C. Gall, "Evoclocks: Software evolution at a glance," in *Proceedings of VISSOFT 2019 (7th Working Conference on Software Visualization)*. IEEE, 2019.

[9] F. Steinbrückner and C. Lewerentz, "Representing development history in software cities," in *Proceedings of SOFTVIS 2010 (5th International symposium on Software visualization)*, 2010, pp. 193–202.

[10] M. Lanza, "The Evolution Matrix: Recovering software evolution using software visualization techniques," in *Proceedings of FSE 2001 (4th International Workshop on Principles of Software Evolution)*. ACM, 2001, pp. 37–42.

[11] M. D'Ambros and M. Lanza, "Software bugs and evolution: A visual approach to uncover their relationship," in *Proceedings of CSMR 2006 (10th Conference on Software Maintenance and Reengineering)*. IEEE, 2006, pp. 10–pp.

[12] R. Wettel, M. Lanza, and R. Robbes, "Software systems as cities: A controlled experiment," in *Proceedings of ICSE 2011 (33rd International Conference on Software Engineering)*. ACM, 2011, pp. 551–560.

[13] S. Benford, C. Brown, G. Reynard, and C. Greenhalgh, "Shared spaces: Transportation, artificiality, and spatiality," in *Proceedings of CSCW 1996 (Conference on Computer Supported Cooperative Work)*. ACM, 1996, pp. 77–86.

[14] S. K. Card, J. Mackinlay, and B. Shneiderman, *Readings in information visualization: Using vision to think*. Morgan Kaufmann, 1999.

[15] J. Sillito, G. C. Murphy, and K. De Volder, "Questions programmers ask during software evolution tasks," in *Proceedings of FSE 2006 (14th International Symposium on Foundations of Software Engineering)*. ACM, 2006, pp. 23–34.

[16] I. Pollack and L. Ficks, "Information of elementary multidimensional auditory displays," *The Journal of the Acoustical Society of America*, vol. 26, no. 2, pp. 155–158, 1954.

[17] B. H. Repp and A. Penel, "Auditory dominance in temporal processing: New evidence from synchronization with simultaneous visual and auditory sequences." *Journal of Experimental Psychology: Human Perception and Performance*, vol. 28, no. 5, p. 1085, 2002.

[18] D. Burr, M. S. Banks, and M. C. Morrone, "Auditory dominance over vision in the perception of interval duration," *Experimental Brain Research*, vol. 198, pp. 49–57, 2009.

[19] J. Anderson and P. Sanderson, "Designing sonification for effective attentional control in complex work domains," *Proceedings of Human Factors and Ergonomics Society Annual Meeting*, vol. 48, no. 16, pp. 1818–1822, 2004.

[20] W. W. Gaver, "Auditory icons: Using sound in computer interfaces," *ACM SIGCHI Bulletin*, vol. 19, no. 1, p. 74, 1987.

[21] G. Kramer, B. Walker, T. Bonebright, P. Cook, J. H. Flowers, N. Miner, and J. Neuhoff, "Sonification report: Status of the field and research agenda," *Report prepared for the National Science Foundation by members of the International Conference for Auditory Display*, 1999.

[22] M. Wertheimer, "Laws of organization in perceptual forms," in *In W. D. Ellis (Ed.), A source book of Gestalt psychology*. Kegan Paul, Trench, Trubner & Company, 1938, pp. 71–88.

[23] G. Kramer, *Auditory Display: Sonification, Audification, And Auditory Interfaces*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[24] C. Armenti and M. Lanza, "Telling software evolution stories with sonification," in *Proceedings of ICPC 2025 (33rd International Conference on Program Comprehension)*. IEEE, 2025, pp. 398–402.

[25] A. Von Mayrhauser and A. M. Vans, "Program comprehension during software maintenance and evolution," *Computer*, vol. 28, no. 8, pp. 44–55, 1995.

[26] H. A. Simon and A. Newell, "Human problem solving: The state of the theory in 1970," *American psychologist*, vol. 26, no. 2, p. 145, 1971.

[27] S. McIntosh, K. Legere, and A. E. Hassan, "Orchestrating change: An artistic representation of software evolution," in *Proceedings of CSMR-WCRE 2014 (Conference on Software Maintenance, Reengineering, and Reverse Engineering, Software Evolution Week)*. IEEE, 2014, pp. 348–352.

[28] T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri, "Challenges in software evolution," in *Proceedings of IWPSE 2005 (8th International Workshop on Principles of Software Evolution)*. IEEE, 2005, pp. 13–22.

[29] C. Gote and C. Zingg, "Gambit—An open source name disambiguation tool for version control systems," in *Proceedings of MSR 2021 (International Conference on Mining Software Repositories)*. IEEE, 2021, pp. 80–84.

[30] S. Amreen, A. Mockus, R. Zaretzki, C. Bogart, and Y. Zhang, "ALFAA: Active learning fingerprint based anti-aliasing for correcting developer identity errors in version control systems," *Empirical Software Engineering*, vol. 25, pp. 1136–1167, 2020.

[31] S. Campanella and M. Lanza, "Hidden in the code: Visualizing true developer identities," in *Proceedings of VISSOFT 2024 (Working Conference on Software Visualization)*. IEEE, 2024, pp. 24–35.

[32] E. Friauf, "Hearing," *e-Neuroforum*, vol. 5, no. 3, pp. 51–52, 2014.

[33] A. H. Caudwell, "Gource: visualizing software version control history," in *Proceedings of OOPSLA 2010 (International Conference Companion on Object Oriented Programming Systems Languages and Applications companion)*, 2010, pp. 73–74.

[34] D. Moody, "The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 756–779, 2009.

[35] T. Ball and S. G. Eick, "Software visualization in the large," *Computer*, vol. 29, no. 4, pp. 33–43, 1996.

[36] C. M. Taylor and M. Munro, "Revision towers," in *Proceedings of VISSOFT 2002 (1st International Workshop on Visualizing Software for Understanding and Analysis)*. IEEE, 2002, pp. 43–50.

[37] D. Rozenberg, I. Beschastnikh, F. Kosmale, V. Poser, H. Becker, M. Palyart, and G. C. Murphy, "Comparing repositories visually with repograms," in *Proceedings of MSR 2016 (13th International Conference on Mining Software Repositories)*. ACM, 2016, pp. 109–120.

[38] M. Lanza and S. Ducasse, "Understanding software evolution using a combination of software visualization and software metrics," *Obj. Logiciel Base données Réseaux*, vol. 8, no. 1-2, pp. 135–149, 2002.

[39] M. Lanza, S. Ducasse, H. C. Gall, and M. Pinzger, "CodeCrawler: An information visualization tool for program comprehension," in *Proceedings of ICSE 2005 (27th International Conference on Software Engineering)*. ACM, 2005, pp. 672–673.

[40] E. Aghajani, A. Mocci, G. Bavota, and M. Lanza, "The code time machine," in *Proceedings of ICPC 2017 (25th International Conference on Program Comprehension)*. IEEE, 2017, p. 356–359.

[41] Y. Kim, J. Kim, H. Jeon, Y.-H. Kim, H. Song, B. Kim, and J. Seo, "Githru: Visual analytics for understanding software development history through git metadata analysis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 656–666, 2020.

[42] J. Ratzinger, M. Fischer, and H. Gall, "EvoLens: Lens-view visualizations of evolution data," in *Proceedings of IWPSE 2005 (8th International Workshop on Principles of Software Evolution)*. IEEE, 2005, pp. 103–112.

[43] M. D'Ambros, M. Lanza, and M. Lungu, "The Evolution Radar: Visualizing integrated logical coupling information," in *Proceedings of MSR 2006 (International workshop on Mining Software Repositories)*. ACM, 2006, pp. 26–32.

[44] R. Wettel and M. Lanza, "CodeCity: 3D visualization of large-scale software," in *Proceedings of ICSE 2008 (30th International Conference on Software Engineering)*. ACM, 2008, pp. 921–922.

[45] R. Wettel, M. Lanza, and R. Robbes, "Software systems as cities: A controlled experiment," in *Proceedings of ICSE 2011 (33rd International Conference on Software Engineering)*. ACM, 2011, pp. 551–560.

[46] M. Bétrancourt and B. Tversky, "Effect of computer animation on users' performance: A review," *Le travail humain*, vol. 63, no. 4, p. 311, 2000.

[47] D. Beyer and A. E. Hassan, "Animated visualization of software history using evolution storyboards," in *Proceedings of WCRE 2006 (13th Working Conference on Reverse Engineering)*. IEEE, 2006, pp. 199–210.

[48] ——, "Evolution storyboards: Visualization of software structure dynamics," in *Proceedings of ICPC 2006 (14th International Conference on Program Comprehension)*. IEEE, 2006, pp. 248–251.

[49] M. F. Kleyn and P. C. Gingrich, "GraphTrace–Understanding object-oriented systems using concurrently animated views," in *Proceedings of OOPSLA 1988 (Object-Oriented Programming Systems, Languages and Applications)*. ACM, 1988, pp. 191–205.

[50] G. Occhipinti, C. Nagy, R. Minelli, and M. Lanza, "SYN: Ultra-scale software evolution comprehension," in *Proceedings of ICPC 2023 (31st International Conference on Program Comprehension)*. IEEE, 2023, pp. 69–73.

[51] J. T. Stasko, "Tango: A framework and system for algorithm animation," *ACM SIGCHI Bulletin*, vol. 21, no. 3, pp. 59–60, 1990.

[52] L. Végh and V. Stoffová, "Algorithm animations for teaching and learning the main ideas of basic sortings," *Informatics in Education*, vol. 16, no. 1, pp. 121–140, 2017.

[53] C. Armenti and M. Lanza, "Using animations to understand commits," in *Proceedings of ICSME 2024 (40th International Conference on Software Maintenance and Evolution)*. IEEE, 2024, pp. 660–665.

[54] ——, "Using interactive animations to analyze fine-grained software evolution," in *Proceedings of VISSOFT 2024 (12th Working Conference on Software Visualization)*. IEEE, 2024, pp. 36–47.

[55] P. Janata and E. Childs, "Marketbuzz: Sonification of real-time financial data," in *Proceedings of ICAD 2004 (9th International Conference on Auditory Display)*, 2004.

[56] A. Walker and S. Brewster, "Spatial audio in small screen device displays," *Personal Technologies*, vol. 4, no. 2-3, pp. 144–154, 2000.

[57] S. Boccuzzo and H. C. Gall, "Software visualization with audio supported cognitive glyphs," in *Proceedings of ICSME 2008 (24th International Conference on Software Maintenance)*. IEEE, 2008, pp. 366–375.

[58] K. Hussein, E. Tilevich, I. I. Bukvic, and S. Kim, "Sonification design guidelines to enhance program comprehension," in *Proceedings of ICPC 2009 (17th International Conference on Program Comprehension)*. IEEE, 2009, pp. 120–129.

[59] C. D. Wickens and Y. Liu, "Codes and modalities in multiple resources: A success and a qualification," *Human Factors*, vol. 30, no. 5, pp. 599–616, 1988.

[60] S. A. Brewster, "Using non-speech sound to overcome information overload," *Displays*, vol. 17, no. 3-4, pp. 179–189, 1997.

[61] D. H. Sonnenwald, B. Gopinath, G. O. Haberman, W. M. Keese, and J. S. Myers, "InfoSound: An audio aid to program comprehension," in *Proceedings of HICSS 1990 (23rd Hawaii International Conference on System Sciences)*, vol. 2. IEEE, 1990, pp. 541–546.

[62] C. J. DiGiano and R. M. Baecker, "Program auralization: Sound enhancements to the programming environment," in *Proceedings of GI 1992 (18th Conference on Graphics, Visualization & HCI)*. Morgan Kaufmann Publishers Inc., 1992, pp. 44–52.

[63] P. Vickers and J. Alty, "CAITLIN: A musical program auralisation tool to assist novice programmers with debugging," in *Proceedings of ICAD 1996 (2nd International Conference on Auditory Display)*. Xerox PARC, Palo Alto, 1996.

[64] K. J. North, S. Bolan, A. Sarma, and M. B. Cohen, "GitSonifier: Using sound to portray developer conflict history," in *Proceedings of ESEC/FSE 2015 (10th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering)*. ACM, 2015, pp. 886–889.

[65] L. I. Berman and K. B. Gallagher, "Using sound to understand software architecture," in *Proceedings of SIGDOC 2009 (27th International conference on Design of communication)*. ACM, 2009, pp. 127–134.